HDNA : Heuristic DNA Computing Algorithm

Dr. Ahmed Tariq Sadiq*& Hasanen Samir Abdullah*

Received on:14/9/2008
Accepted on:31/12/2008

Abstract

The proposed system is based on embedded the heuristic search in DNA search algorithm so to make it more efficient and flexible. The HDNA system is constructed to improve the work of the DNA computing algorithm and enhance the measurement criteria of it by reducing the run time and the memory capacity as well as the number of generated random solutions (strands or states) that are needed to implement the computing algorithm. The experimental results appear that the HDNA using A* and Alpha-Beta is more efficient than using A* and Alpha-Beta each alone.

Keywords : DNA Computing, Heuristic Methods, A* Algorithm, Alpha-Beta Algorithm.

خوارزمية حسابات الحامض النووي التنقيبية الموجهه

الخلاصة

النظام المقترح أعتمد على أساس تضمين البحث التنقيبي الموجه في خوارزميات البحث للحامض النووي، وذلك لجعله اكثر كفاءة ومرونة وقد سمي اختصاراً (HDNA). وهذا النظام المقترح بني لتطوير العمل في حسابات خوارزمية الحامض النووي وكذلك في نحسين المعايير القياسية له من خلال تقليص وقت تنفيذ الخوارزمية وسعة الذاكرة المستخدمة في العمل اضافة الى تحديد عملية توليد الحلول العشوائية (أشرطة الحامض النووي) الاولية والتي نحتاجها لتنفيذ الخوارزمية. النتائج المستخلصة من تجارب النظام المقترح أظهرت ان تطوير خوارزمية الحامض النووي باستخدام خوارزمية A* و خوارزمية Alpha-Beta معاً هو أكثر كفاءة من أستخدام كل خوارزمية على أنفراد.

1. Introduction

There are two fundamentally major approaches in the field of AI. One is often termed traditional symbolic AI, which has been historically dominant. It is characterized by a high level of abstraction and a macroscopic view. Classical psychology operates at a similar level. Knowledge engineering systems and logic programming fall in this category. Symbolic AI covers areas such as knowledge base systems, logical reasoning, symbolic machine learning, search techniques, and natural language processing. The second approach is based on low level, microscopic biological models, similar to the emphasis of physiology or genetics. Neural networks, genetic algorithms and DNA computing are the prime examples of this latter approach. These biological models do not necessarily resemble their original biological counterparts. However, they are evolving areas from which many people expect significant practical applications in the future. DNA computing has been extended in their ways of representing solutions and performing basic processes. A border definition of DNA computing, sometimes called evolutionary computing, includes not only generic genetic information but also some aspects of Artificial Life. Other related areas include evolvable hardware, evolutionary robotics, ant colony optimization and swarm intelligence **[1]**.

2. DNA Computing

DNA is the basic medium of information storage for all living cells. It contains and transmits the data of life for billions of years. Roughly 10 trillion

* **Computer Science Department, University of Technology/ Baghdad**

DNA molecules could fit into a space the size of a marble. Since all these molecules can process data simultaneously, you could theoretically have 10 trillion calculations going on in a small space at once **[2]**. DNA computing began in 1994 when Leonard Adleman has first shown that computing can be done using DNA also, without using usual machine but using test tubes etc. in biological laboratory **[3]**. In the language of computer scientists, 8 binary bits correspond to 1 byte. As the DNA bases are 4 (C, G, A or T instead of 0 or 1), DNA requires half the amount of base pairs, i.e. 4, instead of 8 binary bits, to make one "genetic byte". In DNA computing, also known as molecular computing, a DNA computer is basically a collection of specially selected DNA strands whose combinations will result in the solution to some problem, depending on the problem at hand, technology is currently available both to select the initial strands and to filter the final solution **[4]**.

The fundamental schema of a DNA algorithm, for solving an instance of a combinatorial problem, is the following [5]:

i) Generation of a pool DNA strands encoding all possible solutions (the solution space).

ii) Extraction of those that are the true solutions of the given instance.

The second step is performed by a sequence of elementary extraction sub-steps, where at each sub-step all the strands where a specific sub-strands occurs are selected from the pool and constitute the input for the next extraction sub-step. These two steps are usually of complexity that is linear in time with respect to the size of the given instance. This is the conceptual strength of DNA computing **[5].**

There are some available steps (materials and bio-laboratory techniques) that have been used to build the DNA based computing models. These steps are essentially the following:

1. **Watson-Crick pairing.** Every strand of DNA has its Watson-Crick complement. As it happens, if a molecule of DNA in solution meets its complement, then the two strands will anneal **[4]**.
2. **Polymerases.** Polymerases copy information from one molecule into another **[4]**.
3. **Ligase.** Ligases bind molecules together. For example, DNA ligaes will take two strands of DNA in proximity and covalently bond them into a single strand **[4]**.
4. **Restriction Enzymes (Nucleases).** Nucleases cut nucleic acids; any double stranded DNA that contains the restriction site with in its sequence is cut by the enzyme at that point **[4]**.
5. **Polymerase Chain Reaction (PCR) [11]**

- Polymerase Chain Reaction
- Amplifies (produces identical copies of) selected DNA molecules.
- Makes 2n copies (n: number of iteration)
- Solution filtering or amplification step.

6. **Gel Electrophoresis [4].**

-Separates RNA, DNA and oligonucleotides by length.
- Nucleic acids are mixed with porous gel.
- Molecules can be seen through staining or other methods.
- Electrophoresis purifies molecules **[11]**.

7. **DNA synthesis.** It is now possible to write a DNA sequence on a piece of paper, send it to commercial synthesis facility and receive a test tube

containing the written sequence of DNA**[4]**.

3. Heuristic Search Algorithms

We can saw that many of the problems that fall within the purview of artificial intelligence are too complex to be solved by direct techniques; rather they must be attacked by appropriate search methods armed with whatever direct techniques are available to guide the search. These methods are all varieties of heuristic search. They can be described independently of any particular task or problem domain. But when applied to particular problems, their efficacy is highly dependent on the way they exploit domain-specific knowledge since in and of themselves they are unable to overcome the combinatorial explosion to which search processes are so vulnerable. For this reason, these techniques are often called weak methods. Although a realization of the limited effectiveness of these weak methods to solve hard problems by themselves has been an important result that emerged from the last decades of AI research, these techniques continue to provide the framework into which domain-specific knowledge can be placed, either by hand or as a result of automatic learning **[6]**.

3.1 A* Search Algorithm

The A* algorithm to be discussed shortly is a complete realization of the best first algorithm that takes into account these issues in detail. The following definitions, however, are required for representing the A* algorithm. These are in order.

Definition 1: A node is called open if the node has been generated and the h'(x) has been applied over it but it has not been expanded yet.

Definition 2: A node is called closed if it has been expanded for generating offsprings.

In order to measure the goodness of a node in A* algorithm, we require two cost functions:

- **Heuristic cost.**
- **Generation cost.**

The heuristic cost measures the distance of the current node x with respect to the goal and is denoted by h(x). The cost of generating a node x, denoted by g(x), on the other hand measures the distance of node x with respect to the starting node in the graph. The total cost function at node x, denoted by f(x), is the sum of g(x) plus h(x) **[7]**.

Here are the basic steps that are considered to implement the A* procedure to solve problems in an intelligent manner **[8]**:

1. Operations on states generate children of the state currently under examination.

2. Each new state is checked to see whether it has occurred before thereby preventing loops.

3. Each state n is given an f value equal to the sum of its depth in the search space g(n) and a heuristic estimate of its distance to a goal h(n).

4. States on open are sorted by their f examined or a goal.

5. As an implementation point, the algorithm can be improved through maintenance of perhaps heaps or leftist trees.

3.2 The Alpha Beta Pruning

We will discuss a special type of algorithm, which does not require expansion of the entire space exhaustively. This algorithm is referred to as alpha-beta cutoff algorithm. In this algorithm, two extra plies of movements are considered to select the current move from alternatives. Alpha and Beta denote two cutoff levels associated with MAX and MIN nodes. As it is

mentioned before the Alpha value of MAX node cannot decrease, whereas the Beta value of the MIN nodes cannot increase. But how can we compute the Alpha and Beta values? They are the backed up values up to the root like MINIMAX. There are a few interesting points that may be explored at this stage. Prior to the process of computing MAX / MIN of the backed up values of the children, the Alpha-Beta cutoff algorithm estimates e(n) at all fringe nodes n. Now, the values are estimated following the MINIMAX algorithm. Now, to prune the unnecessary paths below a node, check whether **[7]**:

- the Beta value of any MIN node below a MAX node is less than or equal to its Alpha value. If yes, prune that path below the MIN node.
- the Alpha value of any MAX node below a MIN node exceeds the Beta value of the MIN node. If yes, prune the nodes below the MAX node.

Based on the above discussion, we now present the main steps in the α-β search algorithm.

4. HDNA: The Heuristic DNA Computing

The new idea depends on the fact that says "heuristic search studies the methods and rules of discovery and invention", now it is the time to discover and invent more efficient and stronger computing fashion. The idea suggests embedding the heuristic search in DNA search algorithm to make it more efficient and flexible, we can benefit from the heuristic search properties to improve the work of the DNA computing algorithms to get the same results that appear when the DNA computing algorithm is run alone, but in less run time required and the space area of the computer memory that is required to implement the algorithm. The main two heuristic search properties that are depended on in this work are as follows:

- Problem Reduction: very useful aim in intelligent search because reducing the problem state space (according to the heuristic function) makes the search faster and more desired.
- Guarantee of Solution: another useful aim in intelligent search, is the intelligent search algorithm must give a guarantee to find the best solution(s).

Here the system benefits from the last two parts by appending the benefits of the Alpha Beta pruning and the A* technique, as mentioned before, Alpha Beta pruning will reduce the problem search space as much as possible; and the A* technique will guarantee to get solution(s), thus when they are applied together in one implementation phase, of course the results of the system are obtained with the following properties:

1. Problem Reduction, this property is obtained in two ways, the main one from applying the pruning technique in the Alpha Beta algorithm, while the other one from storing process, after generating the random solutions (strands), which store the resulting paths as an A* tree structure.
2. Guarantee of solution, this property appears clearly with applying the A* search algorithm to those solutions (strands) that are stored in the A* tree structure, because the A* search technique evaluates each node in the tree by the evaluating function f(n) which is calculated as the addition of the generation function g(n) to the heuristic function h(n): $f(n) = g(n) + h(n)$.
3. Reduce the run time, according to the above two properties the run time is reduced to very acceptable ratio in comparison with run time calculated from

applying the DNA search algorithm alone.

4. Reduce the amount of needed memory capacity, the process of heuristically generating random solutions and finding the target (desired) solutions makes the HDNA need less memory capacity amount.

In the block diagram below there are two modification steps, the first modification is obtained from (Alpha Beta Pruning) while the other one from (A* search technique) and then implement them together. Figure (1) represents the system architecture of the HDNA system.

The below algorithm illustrates the general algorithm of the HDNA system.

Procedure: Preliminary Process
Input: Graph G with V states
Output: 2D matrix with Preliminary Process File F
/*this file contains Watson-crick pairing & restriction enzyme to each state*/
Begin
For each state Vi in the Graph
Make the Watson-crick pairing
For each Watson-crick pairing(Vi)
Begin
Make the ligase operation
If Vi has a connection link to another Vj then
Determine the cut point, K
Do restriction enzymes between Vi & Vj in the K point
Store a value of one in 2D matrix in the location ij
Else store a value of zero in 2D matrix in the location ij
End;
Store the Watson-crick and restriction enzyme results in file F
End.

Procedure: Alpha Beta pruning to Generate Random Solutions (Paths)
Input: 2D matrix with Preliminary Process File F, N
/* N= maximum strand length */
Output: Tree contains Generating Solutions (strands)
Begin
Labeled the states as 1,2,3,…k in ascending order of degree
d(1)>= d(2)>=…..>=d(k)
Give each state in the graph (except the desired state(s)) a min value called it Beta-Value.
Give the desired state(s) a max value called it Alpha-Value.
For each state u=1,2,3,…..k in turn
/* Initialization*/
Begin
Select the random state Sr
If Sr(value) = Beta-Value then Desired = False
End;
Else
Begin
Sr(value) = Alpha-Value
/* desired = True */
Repeat
/* For each unvisited neighbor W of Sr (Iteration) */
Compute the number of unvisited neighbors of W
Select Sr+1 state
If there are many possible choices then select W with the smallest label
Until the last selected state has no unvisited neighbors or length of the solution (strand) = N
Store the resulted strand in an A* tree structure
End;
End.

/* The result: solutions (strands) visited states U= Sr,…..,Sk such that Sk has no unvisited neighbors or length of the solution (strand) = N stored as a tree data structure */

Procedure: A*

Input: Tree contains solutions (strands), I.S, G.S /* where I.S = Initial State & G.S = Goal State */
Output: Tub(b) array (Pa* solutions (strands))
Begin
b = 0
Open = [I.S]
Closed = []
While open <> [] do
Begin
Remove the leftmost state from open, call it X
If X = G.S then
Begin
Return the Path from I.S to X
Tub(b) = Path
b = b + 1
End;
Else begin
Generate children of X
For each child of X do
Begin
Assign the child a heuristic value f(n)
Add the child to open
End;
Put X on closed
Re-order states on open by heuristic merit f(n) (best leftmost)
End;
Return fail
End;
End.
/* The result: solutions (strands) that start and end with predefined I.S and G.S respectively */

Procedure: Gel electrophoresis
Input: Tub(b), Pa* solutions (strands) that were stored in Tub array, M
/* where M = determined strand length */
Output: strand(c) array (Pgel solutions (strands))
Begin
c = 0
For i = 1 to b (Tub length)
Begin
If Tub(i).length = M (or >= M) then
Strand(c) = Tub(i)
c = c + 1
End;
End.
/* The result: solutions (strands) with determined length by factor M */

Procedure: Synthesis
Input: Strand(c) array (Pgel solutions (strands)) that were stored in Strand array
Output: Tub(index) array (Ps final solutions (strands))
Begin
index = 0
For i = 1 to c (Strand length)
Begin
Path = Strand(i)
For j = 1 to Path.length-1
S = Path(j)
For k = j+1 to Path.length
begin
If S = path(k) then
Repetition = True
Else /* Repetition = False */
Tub(index) = Path
index = index + 1
End;
End;
End.
/* The result: final solutions (strands) or desired solutions (paths) */

Main: HDNA
Input: Graph G with S states
Output: Tub(index) array (final solutions (strands))
Begin
Preliminary Process;
Alpha Beta Pruning to Generate Random Solution (strands);
A*;
Gel electrophoresis;
Synthesis;
End.

5. Case Studies

The HDNA system is constructed to improve the work of the DNA computing algorithm and enhance the measurement criteria of it by reducing

the run time and the memory capacity as well as the number of generated random solutions (strands or states) that are needed to implement the computing algorithm. To prove that the HDNA system will work correctly, it is considered very important matter to test it with several applications. The applications (we call each one a case testing) that we selected are the following:

1- The Traveling Salesman Problem (TSP).

2- The Routing with Minimum Cost Problem (RMCP).

5.1 TSP in HDNA

The traveling salesman problem is chosen as a benchmark. The traveling salesman problem is to find an acceptable path(s) for a given set of vertices (cities) and edges (links). In addition, the solution path must contain all the cities given, each only once, and begins from the specified city to which the tour ends **[9]**.

The TSP will be implemented via HDNA system. Resulting paths will remain the same as those appear in implementing DNA algorithm alone while the measurement criteria (G.R.P, PCR.P, R.T, and M.C) are changed to better. Here we present practical results that are obtained from applying the various approaches of the HDNA proposed method through the TSP with various cases. Figure (2) represents the run time comparison of implementing the TSP via DNA, DNA with ABP, DNA* and HDNA respectively, while Figure (3) represents the memory capacity comparison from the results of TSP.

5.2 RMCP in HDNA

The following sections give a description of two routing heuristics developed for the HDNA system. Both routing techniques are routed via DNA computing algorithm. The two routing techniques can be characterized as follows **[10]**:

1. Classical Router - allows unassigned graph nodes, which is called Routing Problem (or simply RP).
   - Tour generated is always feasible, that is, all nodes will be visited after starting with initial node without interesting with target node which one is.
   - Requests that cannot be routed will be left disconnected since routing the request may result in an infeasible tour (there is no completed routing tour).
2. Optimal Router - allows assigned graph nodes, it is called Routing with Minimum Cost Problem (or simply RMCP).
   - Tour generated may be feasible, that is, some of the resulting paths (at least one) may be the optimal ones.
   - No response to the requested path because there is no routing path or there are no weighted links corresponding to request value.

The RMCP will be implemented via HDNA system. Resulting paths will remain the same as those appear in implementing DNA algorithm alone, while the measurement criteria (G.R.P, PCR.P, R.T, and M.C) are changed to better. Here we present practical results that are obtained from applying the various approaches of the HDNA proposed system through the RMCP with various cases. Figure (4) represents the run time comparison of implementing the HPP via DNA, DNA with ABP, DNA* and HDNA

respectively, while Figure (5) represents the memory capacity comparison from the results of RMCP.

6. Conclusions

The following items represent the important conclusions which are drawn from the Heuristic DNA System work:

1- The HDNA system reduces the run time to less than half time that DNA computing algorithm needs to implement alone and reduces the need for the memory capacity to one to seven (1/7 approximately) over the standard DNA algorithm needs.

2- The using of the Alpha Beta Pruning limits the generating of random solutions (strands), the ABP part is more efficient in reducing the run time. On the other hand because of using tree structure to store the solutions (strands) as a result of the generation process, the A* search algorithm is more efficient in reducing the need for the memory capacity.

3- The heuristic search (Alpha-beta and A*) makes the randomness (G.R.P) in the DNA computing algorithm informed randomness (guiding the G.R.P stage) by controlling the generation process to give less preliminary solutions (strands) in order to reduce the random search space with guarantee to get the same final solutions (strands) than if the DNA computing is run alone.

4- One of the difficulties that has faced this work is the nature of used graph(s) in each case testing, this means that the run time and the needed memory capacity will increase in either the selected graph nodes increase or the complexity of the graph increase (the graph has more links or connections between its nodes). Another difficult case faced the HDNA system, is the use of the A*-search algorithm which needs a tree structure to work on it, and so to give appropriate results, thus the work requires replacing the method that is used to store the generated random solutions (strands) into tree structure as an alternative storing method to the array structure. When this job is achieved the conversion process adds to the HDNA system a new powerful property.

References

[1]. T. Munakata, ***"Fundamentals of the New Artificial Intelligence"***, Second Edition, Springer, 2008.

[2]. ***"DNA Computing, State of the Art"***, CPSC 601.73 http://www.cpsc.ucalgary.ca/~omair/cpsc60173/presentation, 28-1-2003.

[3]. G. P. Raja Sekhar, ***"DNA Computing-Graph Algorithms"***, The Indian Programmer, Supported by Com MaC-KOSEF, Korea, 2003.

[4]. T. Zingel, ***"Formal Models of DNA Computing: A survey"***, Proc. Estonia Acad. Sci. Phys. Math., 49, 2, 90-99, 2000.

[5]. G. Franco, C. Ciagulli, C. Laudanna and V. Manea, ***"DNA Extraction by Cross Pairing PCR"***, Springer, 2005.

[6]. M. Bramer and V. Devedzic, ***"Artificial Intelligence Applications and Innovations"***, Kluwer Academic Publishers, Springer science + Business Media, Inc., 2004.

[7]. A. Konar, ***"Artificial Intelligence and Soft Computing"***, Behavior and

Cognitive Modeling of the human brain, CRC Press, 1999.

[8]. G. F. Luger, ***"Artificial Intelligence Structures and Strategies for Complex Problem Solving"***, Fourth Edition, Pearson Education Asia Ltd., 2002.

[9]. J. Y. Lee, S. Y. Shin, S. J. Augh, T. H. Park and B. T. Zhang, ***"Temperature Gradient-Based DNA Computing for Graph Problems with Weighted Edges"***, Springer, 2003.

[10]. S. E. Chan, ***"Meta Heuristics for Solving the Dial-A-Ride Problem"***, Ph. D. Thesis, North California State University, USA, 2004.

[11]. L. M. Adleman, ***"Computing with DNA"***, Magazine: Scientific American, 1998.

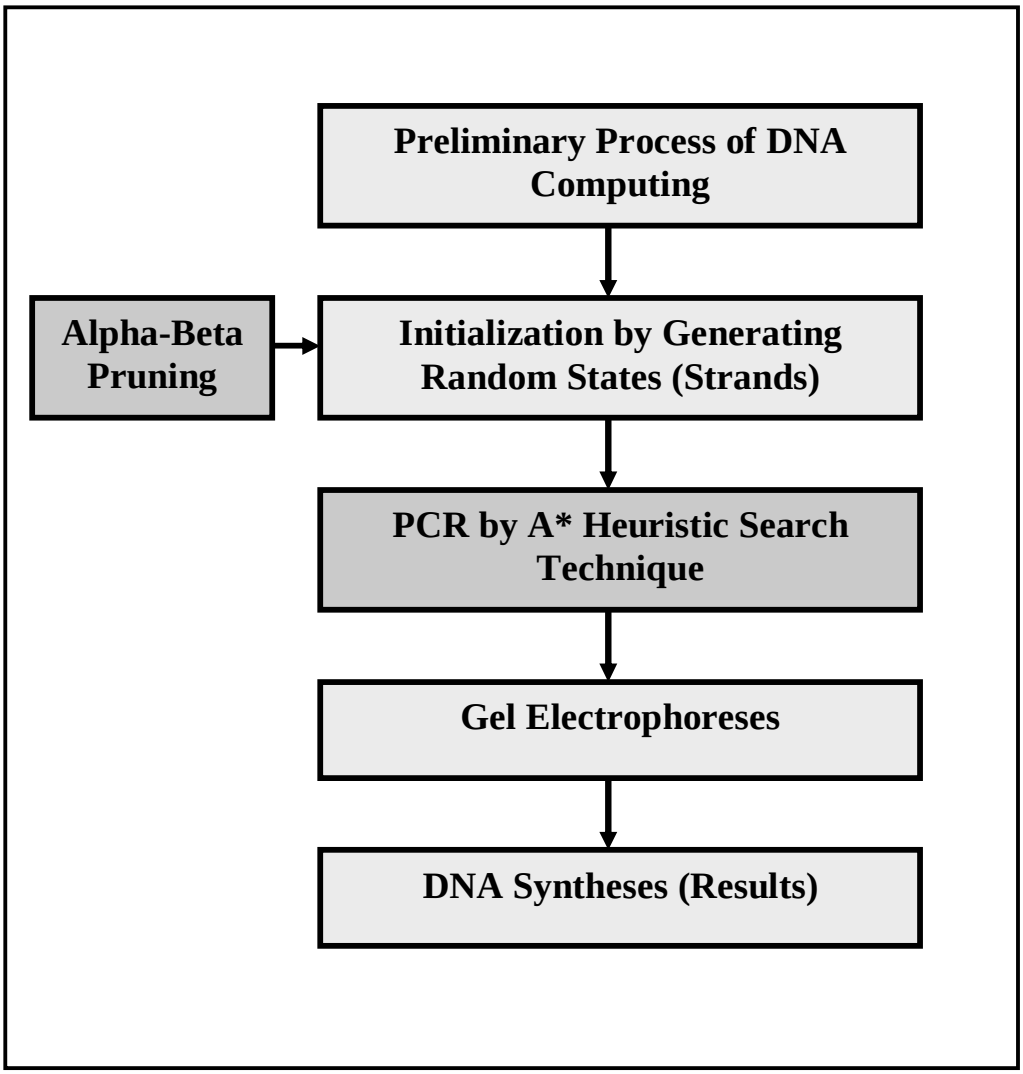

Figure (1) System Architecture of HDNA

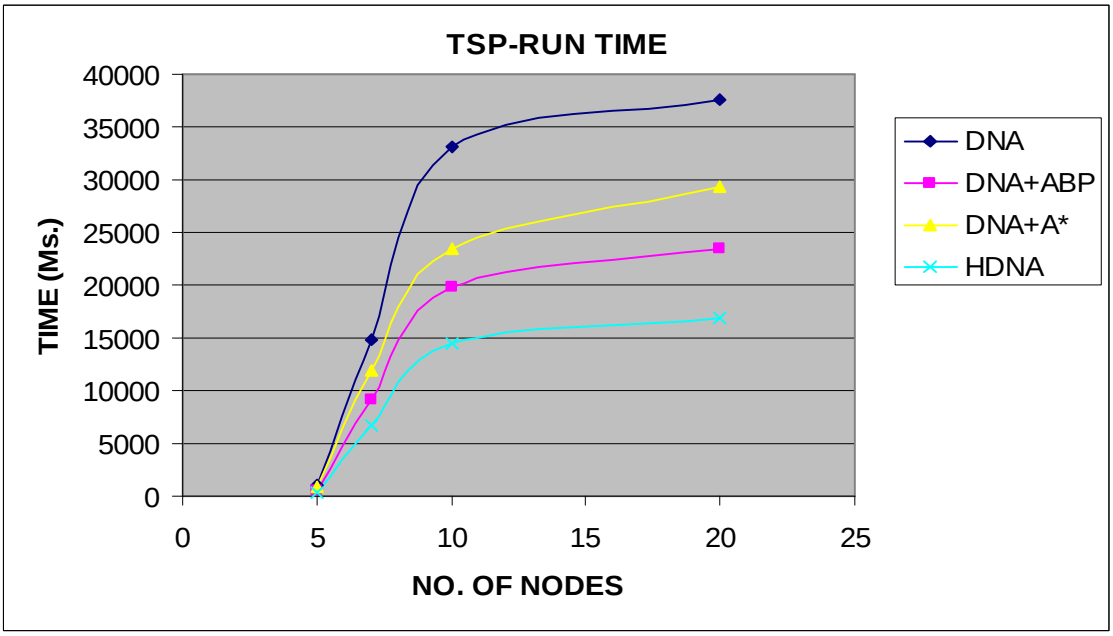

Figure (2) Run Time of TSP via Various HDNA Approaches

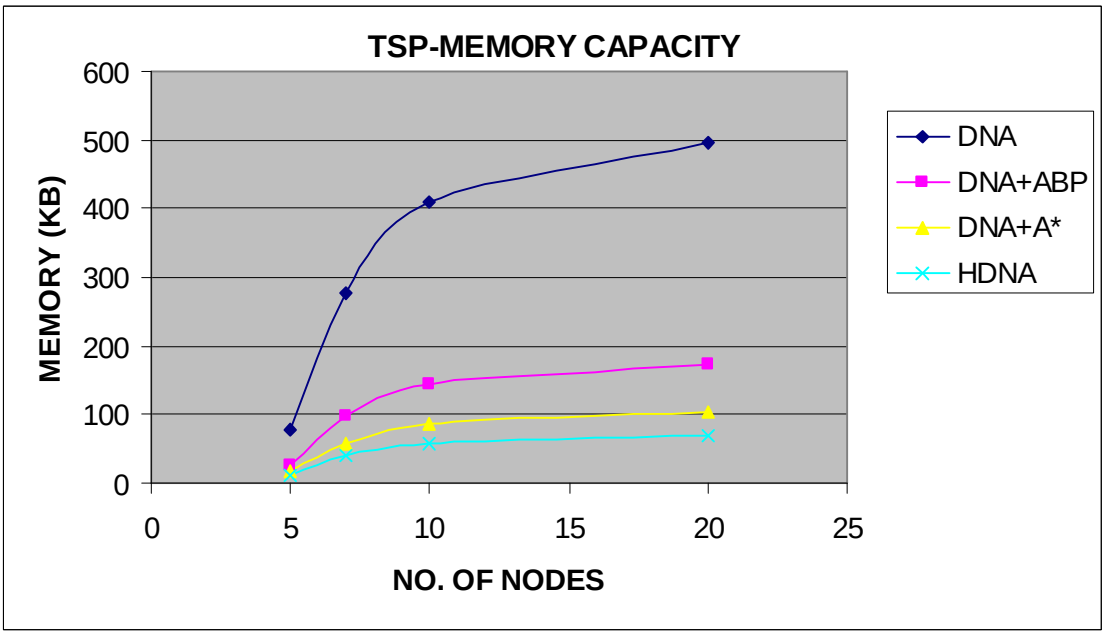

Figure (3) Memory Capacity of TSP via Various HDNA Approaches

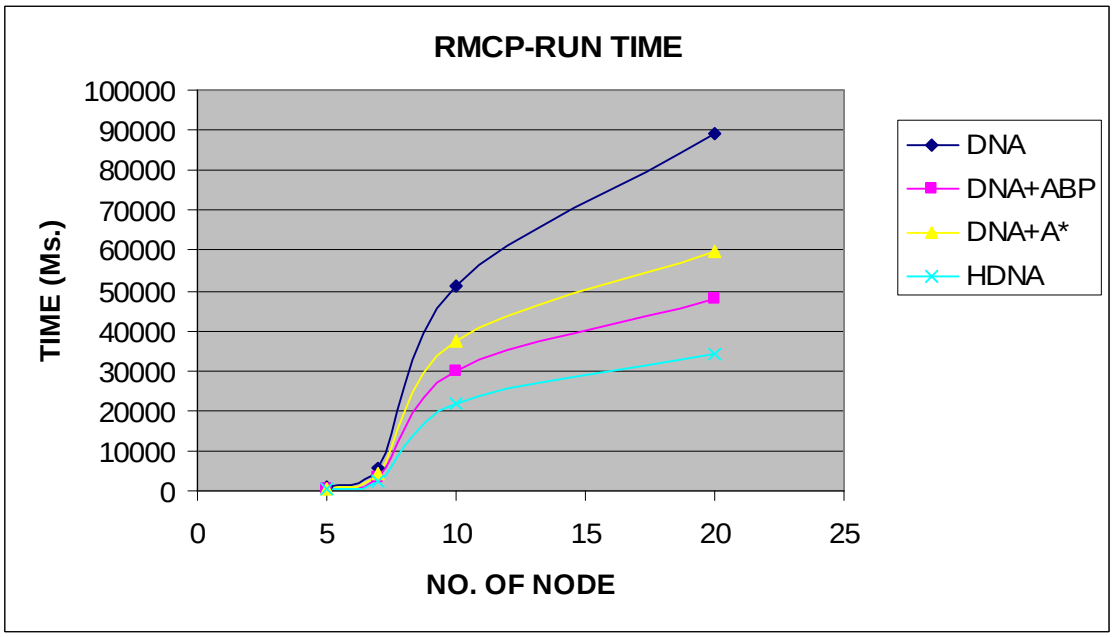

Figure (4) Run Time of RMCP via Various HDNA Approaches

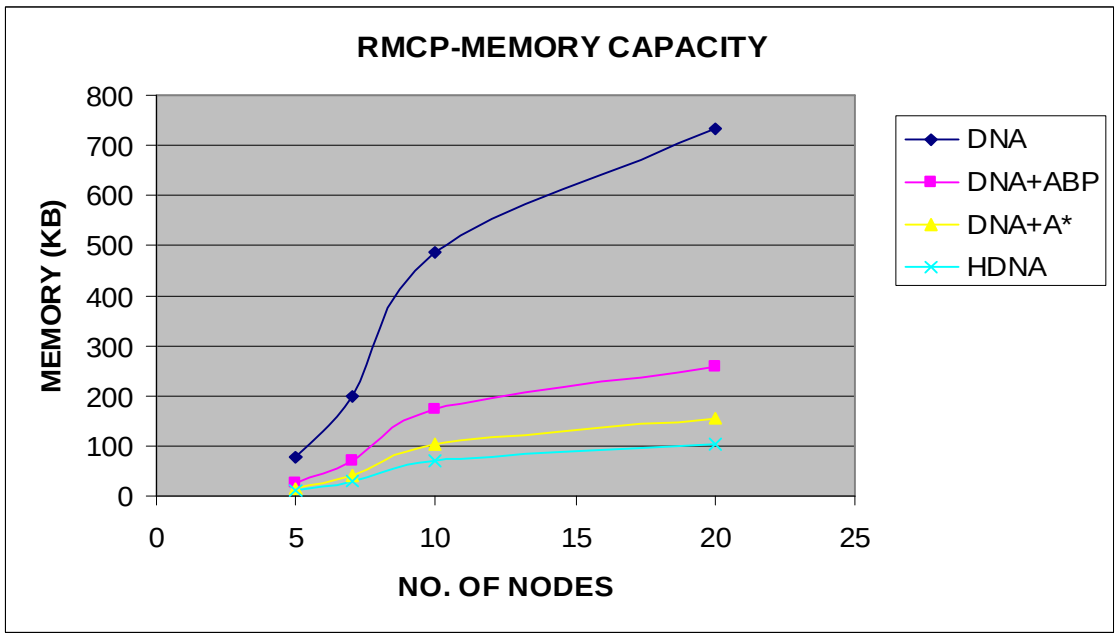

Figure (5) Memory Capacity of RMCP via Various HDNA Approaches